AN INTEGER PROGRAMMING APPROACH FOR ASSIGNING VOTES IN A DISTRIBUTED SYSTEM

by

D. VENKAIAH

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

APRIL, 1992

An Integer programming Approach for Assigning votes in a Distributed System

A Thesis Submitted

in Partial Fulfilment of the Requirements

for the Degree of

MASTER OF TECHNOLOGY

by

D. Venkaiah

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

April, 1992

16/4/92

CERTIFICATE

This is to certify that the work entitled "AN INTEGER PROGRAMMING APPROACH FOR ASSIGNING VOTES IN A DISTRIBUTED SYSTEM" by Dukkipati Venkaiah has been carried out under my supervision and has not been submitted elsewhere for a degree.

Pankaj Jalote

(Dr. Pankaj Jalote)

Assistant Professor

Dept. of Computer Science and Engg.

I.I.T. Kanpur

Kanpur

April, 1992

ACKNOWLEDGEMENTS

I would like to thank my guide and mentor Dr. Pankaj Jalote for introducing me to the field of fault-tolerant computing and for suggesting such an interesting problem to work on. His valuable advices and constant encouragement given at every stage, have a great influence in shaping this work. It was a rewarding experience to work with him and I very much thank him for the time he has spent with me whenever I wanted to discuss about the thesis work.

I would like to thank all my classmates, in particular Srivakumar, manoj, barraju, surendra, venumadhav, bhargava and Ghatare, for the interesting discussions we had about academic and nonacademic matters. I would like to express my gratitude to siva, for clarifying the integer programming concepts and for the long discussions we had about the formulations. I would like to thank ashish for helping me to learn LaTex.

I would like to thank all A-top and H-top residents of Hall IV, in particular Vishu, LP, shankar, karra, umakanth, subba and ANJI, for the scintillating discussions, funny Phatta matches and movie shows which made my stay here more enjoyable.

I would like to thank my parents for the patience they have shown in the course of this work. Last but by no means the least, I would like to express my sincere gratitude to my brother who gave me the motivation to go for higher studies. Without his encouragement, I might not have started this work.

Abstract

Replication is a well known technique to achieve high availability of data in a distributed system. Due to partition failures, the network may split in to a set of groups. The transactions getting performed simultaneously in these groups might cause inconsistency of replicated data. Voting is a commonly used approach to maintain consistency of replicated data. In this approach, each node is assigned a particular number of votes, and any group with majority of votes can perform critical operations. The number of votes assigned to the nodes can have a significant impact on the system performance. In this thesis, we propose an integer programming approach for the vote assignment problem using average transaction gain per unit time as the performance metric. We use monte-carlo simulation to find the most likely groups formed due to partition failures and the average transaction gain in them per unit time. We use these groups and obtain the vote assignment using integer programming. We suggest three integer programming formulations for the vote assignment problem. Unlike the heuristics proposed in the literature, this approach uses global information to determine the vote assignments. We have tried this approach for different networks and it is observed that in many of the cases this approach is assigning votes equivalent to or better than the best vote assignment given by the heuristics. Based on our experiments, we feel that this approach will result in a better vote assignment when the most likely groups formed due to partition failures are few in number.

Contents

Chapter 1

INTRODUCTION

Distributed computer system is characterised by the presence of a number of processing units connected through a communication network. The availability of data in the system depends highly upon the reliability of the site at which it is stored. By replicating critical data at multiple sites, which have independent failure modes, the probability of the data being accessible even in presence of node and link failures can be increased [CD 88, PN 86, PB 85]. The presence of data at multiple sites can also increase the response time of the system.

However, data replication requires that sites coordinate within themselves for accessing the data, such that the overall view of any process is as if there was a single copy in the system. This requires replica control protocols. A replica control mechanism has to ensure that a consistent view of the data is offered to user processes even in the face of node failures and network partitioning.

There are various methods for controlling access to replicated data [DGB 85]. One of the well known strategy is the majority voting [TH 79]. In majority voting, each site is assigned a particular number of votes. A node which wants to perform an update on the data must collect a majority of votes before performing the operation. In otherwords a set of nodes which have majority of votes should agree for this operation to proceed. Since only one of the groups formed during a partition failure can get majority of votes it satisfies the mutual exclusion criterion even under network partitioning.

A generalization of majority voting is the weighted voting approach [GI 79]. In weighted voting a site is permitted to read only if it has obtained a read quorum of r votes and is allowed to write only if it has obtained a write quorum of w votes. For doing any read opeartion the node will read the replicas at all the nodes which have sent these votes and takes the most recent copy (using version vectors [PR 83]) and to update it updates all the replicas present at these nodes. Read quorum (r) and write quorum (w) are such that $w > N/2$ and $r+w > N$, where N is the total number of votes in the system. The above assignment of quorums will make sure that every read quorum and every write quorum will have a non-null intersection and also every write quorum intersects with other write quorums. Non-null intersection between read and write quorums will makes sure that the value read will be the recent one and non-null intersection between write quorums will make sure that mutual exclusion criteria is satisfied. Many variations of the weighted voting technique have been proposed [RT 88, CMM 90, JP 86, JM 87, DD 89, BGA 86].

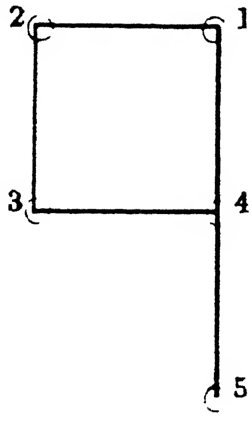


Figure 1.1: Sample network topology

1.1 Need for optimal vote assignment

The performance of the system using weighted voting depends critically on how the votes are assigned. For example, let us consider the network shown in Fig. 1.1.

Let us assume uniform vote assignment (1 vote to each node) and let read quorum = write quorum = 3. When node 3 and link (1,4) fail none of the groups can perform operations. If these two fail very often the system throughput will go down drastically. If node 4 and 5 are relatively more reliable than 1 and 2 then one of the following vote assignments (1,1,1,3,1) or (0,0,0,1,0) could have given better throughput.

Different votes assigned to the nodes will give priorities to the nodes in transaction processing [MM 89]. Giving all votes to a single node (singleton voting) is similar to the well known primary copy [AD 76] approach. Whatever be the vote assignment, it might always lead to a halt state of the system where system can not perform any operation due to non-availability of sufficient votes to obtain a quorum in any of its

groups.

Since halt states will reduce the performance of the system, the vote assignment should be such that it reduces that possibility. If we consider a general network with N nodes the number of possible vote assignments is 2^{N^2} [GB 85]. So it is not feasible to enumerate the vote assignments for big networks. Some other methods are needed for assigning votes to nodes in a network.

1.2 Our approach

In this thesis, we propose an approach for assigning votes, which is based on integer programming. It is an engineering approach, to the vote assignment problem leading to approximate solutions. For a network, first the top k partitions are determined (using simulation), where k is a parameter and is chosen such that most of the likely partitions are included. With these k partitions, the vote assignment problem is formulated as an integer programming problem, with the objective of minimizing the average number of transactions that cannot be satisfied due to partitions. So the metric that we use for evaluation is the average number of transactions "lost" due to partitions per unit time.

The entire procedure has been implemented, in the Vote Assignment Tool (VAT). VAT takes as input the topology of the network, along with failure probabilities. It performs simulation and formulates the problem as an integer programming problem and solves it using known techniques for integer programming. The final output of

VAT is the vote assignment, together with performance data of the vote assignment.

1.3 Outline of the thesis

The rest of the thesis is organised as follows. In the next chapter, we briefly discuss the different types of vote assignment approaches proposed in the literature. In Chapter 3, we discuss our approach in detail. Here we give three integer programming formulations for vote assignment problem. Chapter 4 deals with the implementation details of Vote Assignment Tool (VAT). In chapter 5, we give the experimental results obtained using VAT. In chapter 6, we offer some conclusions.

Chapter 2

RELATED WORK

The group of nodes which should agree for the operation to proceed is called the quorum group for that operation. In weighted voting scheme, quorum groups are formed dynamically by collecting votes from different nodes. There is another approach proposed in [GB 85] called coteries. Here quorum groups (which may be exponential in number) are maintained explicitly. Let U be the set of nodes in the system and C be a set of groups. C is a coterie iff —

1. If $S_i \in C$ then $S_i \neq \emptyset$ and $S_i \subseteq U$.

2. If $S_i, S_j \in C$ then $S_i \cap S_j \neq \emptyset$.

3. If $S_i \in C$ then for all $S_j \subset S_i, S_j \notin C$.

Here for any operation to take place a node should get consent from all nodes present in atleast one of the groups. Since there is a non-null intersection among

the groups in coterie, the mutual exclusion constraint is satisfied. Coterie will give better reliability than voting. As is shown in [GB 85], every vote assignment can be represented by using a coterie but the reverse is not true. It was also proved that getting the best coterie is much more complex than getting the best vote assignment. Enumeration algorithm was proposed in it, which gives all coteries corresponding to vote assignments but only a subset of ND-coteries. This algorithm is not feasible for big networks (>5 nodes). There were several variations of this concept proposed by Maekawa [MK 85], Tripathi [SK 91] et al. But all of these approaches suffer from the drawback that the groups explicitly stored may not be the best one and also there is an overhead involved in maintaining the quorum groups.

Garcia-molina and Barbara [BG 87, BG 84] have proposed a number of heuristics with different types of metrics, for general networks. These heuristics determine the votes to be assigned to a node based on the failure probability of the nodes and the links. They use local information, failure probability of neighbouring nodes/links, in determining vote assignment. For example, in one of the heuristics the votes of a node are proportional to the product of its reliability and sum of reliabilities of all the links incident to it. Tang and Kain [TK 91] have proposed algorithms for optimal vote assignment for networks in which links do not fail. They have used a logarithmic function $\log(\frac{p_i}{1-p_i})$ where p_i is the probability of node i to be up. Since links do not fail, with this vote assignment it can be proved that a group G_i will get majority of votes iff probability of occurence of G_i is greater than probability of occurence of $\overline{G_i}$, which is a key factor in getting optimal vote assignment. They have also proposed a heuristic for general networks. Tang and Natarajan [TN 89] have

proposed an approach to get the optimal acceptance sets (equivalent to coteries) by formulating this as a sparse integer programming problem, but this works only for small systems because for large systems the constraints becomes very large making it very difficult for integer programming solution.

In our approach, we consider static vote assignment for a general network where nodes and links can fail. There were different metrics proposed for evaluating vote assignment, such as probability that the system is not in halt state [BG 87], node vulnerability [BG 84] etc. We consider the metric as the average number of transactions that are lost due to node failures and network partitions per unit time. A request arriving in a partition not having the necessary quorum, is considered as lost.

We take an engineering approach to solving the problem of vote assignment. The top k partitions of the network are considered and then the problem is framed as an integer programming problem, with the goal of minimizing the operations lost. We have designed a tool, called Vote Assignment Tool (VAT), that performs all these steps, and given the network configuration, outputs a possible vote assignment.

proposed an approach to get the optimal acceptance sets (equivalent to coteries) by formulating this as a sparse integer programming problem, but this works only for small systems because for large systems the constraints becomes very large making it very difficult for integer programming solution.

In our approach, we consider static vote assignment for a general network where nodes and links can fail. There were different metrics proposed for evaluating vote assignment, such as probability that the system is not in halt state [BG 87], node vulnerability [BG 84] etc. We consider the metric as the average number of transactions that are lost due to node failures and network partitions per unit time. A request arriving in a partition not having the necessary quorum, is considered as lost.

We take an engineering approach to solving the problem of vote assignment. The top k partitions of the network are considered and then the problem is framed as an integer programming problem, with the goal of minimizing the operations lost. We have designed a tool, called Vote Assignment Tool (VAT), that performs all these steps, and given the network configuration, outputs a possible vote assignment.

Chapter 3

PROBLEM FORMULATION

We model a distributed system as an undirected graph where each node represents a computer and each edge a communication link. Data is replicated at all the nodes present in the network. We assume that there is an underlying concurrency control mechanism which takes care of consistency criteria (such as serializability [BH 87]) during concurrent requests. We assume that all nodes in the network have same processing power.

Both nodes and links can fail. Failures might cause partitioning of the network. We assume that all the failures are statistically independent. It is assumed that all the processors are fail-stop [SS 83], they fail by stopping to function (no byzantine failures [LSP 82]) and there are no transmission errors. We do not distinguish between node failures and network partitions; a node failure appears as a network partition, with one partition having only the failed node. We assume that the life and repair

rates of nodes and links are exponentially distributed. In our model we work with the steady state probabilities of the node/link to be up. We assume that the following parameters are known about the network:

- P_i = steady state probability for the node i to be up
- l_i = steady state probability for the link i to be up
- λ_i = arrival rate of requests at a node i

We consider the average number of transactions that are performed by the system per unit time as the performance criterion. The vote assignment should be such that it maximises the average number of transactions performed by the system per unit time. The basic idea is that when a network gets partitioned, the vote assignment should be such that the group that has the largest number of transactions coming in should get the majority. Since there are many possible partitions, we have to select a vote assignment such that for the majority partitions the sum of the average number of transactions that get performed per unit time is maximum.

3.1 Selecting Network Partitions

Since the reliabilities of the nodes and links is known monte-carlo simulation [ND 79] is performed to see how the network gets partitioned due to these failures. As the arrival rates at the nodes is known, transactions that will get done in each of the groups formed if they were assigned majority of votes can be calculated.

For a group G_i :

Possible transaction gain in G_i per unit time

TR_GAIN_RATE(G_i) = $\sum_{j \in G_i} \lambda_j$

Average possible transaction gain in G_i

AVG_TR_GAIN(G_i) = $(TR\_GAIN\_RATE(G_i) * FT_i)$

where FT_i is the fraction of total time for which group G_i is present.

A simple strategy could have been to find the average possible transaction gain in each of the possible groups and assign votes such that the average transaction gain is maximum. Since the possible set of groups formed due to partition failures is exponential in N, this will lead us to keeping track of exponential set of groups. Hence, it will not be possible to use this approach for large networks.

Whatever be the vote assignment only one of these groups formed at any instance of time can perform restricted operations such as updates. So it will be better if we assign majority of votes to the high priority groups formed during simulation time. We define the highest priority group (HPG) as a group which has maximum potential for transactions that will get done if it was assigned majority of votes. Let C be the set of groups formed due to a partition failure, HPG can be defined as :

HPG $\in$ C such that TR_GAIN_RATE(HPG) $\geq$ TR_GAIN_RATE(G_i) where $G_i \in C$.

In our modelling, we keep track of only the HPG and group next to HPG (NHPG) groups. NHPG $\in C$ such that TR_GAIN_RATE (NHPG) $\geq$ TR_GAIN_RATE (G_i) and NHPG $\neq$ HPG where $G_i \in C$.

We find out the AVG_TR_GAIN(G_i) for all G_i when they are present as HPG or NHPG. Since it is assumed that the big groups formed due to network partitions are less, there won't be much overhead in keeping track of the transactions that will be done in these groups. The groups which are present as NHPG are also considered to take care of a situation in which some groups might be present as one of the high priority groups but they are present as HPG for only a short duration. We have observed that the AVG_TR_GAIN will give better priorities when we consider both HPG, NHPG groups. An ideal case could have been to take AVG_TR_GAIN whenever the group is present. But as it is shown above that it will lead us to keeping track of exponential set of groups.

Through simulation we get a set of groups with possible total transaction gain in each group when it is present as HPG or NHPG.

3.2 Integer programming formulations

It is easy to observe that it may not always be possible to assign majority of votes to all these groups. Now our aim will be to assign votes in such a way that the groups getting majority of votes have maximum sum of AVG_TR_GAIN. We have formulated this as an integer programming problem [HS 89]. As it is clear from the formulations

given below, the number of constraints are proportional to the number of groups for which we are trying majority vote assignment. Integer programming problems complexity is exponential in both the number of variables and number of constraints [KM 84]. So it is not possible to give large set of groups. For our experiments, top 20 groups are given to the integer programming formulation. The number of groups can be changed, however the number of groups considered should not be very high due to the NP-completeness of the integer programming problem. As we have assumed, it may not always be the case that the most likely big groups formed are very few. When the average transactions gain in the groups other than top 20 is less than $\frac{1}{20}th$ of that of the top most group, we assume that they have negligible effect on the average transaction gain in the system. Otherwise omission of the groups other than the top 20 groups might be costly. When this is the case, VAT selects the best one out of singleton or uniform or vote assignment done by integer programming formulation, using average transaction gain in the system per unit time as the metric.

We propose three integer programming formulations for doing this. In formulations 1 and 2 we select the best subset out of these k (k = 20) groups which will get majority votes, such that the sum of average transactions gained in them is maximum. If there are a number of possible vote assignments which can do the same then these formulations will select the one with minimum total votes assigned. In formulation 1 each of the k groups is given as a constraint to the integer programming routine. In formulation 2, first the intersecting groups are found and then integer programming solution is tried. The third formulation uses one of the above two formulations to find the groups which should get majority of votes, if there are a number of vote

assignments possible then it will select the one which assigns votes similar to the one assigned by the modified version of heuristic proposed in [BG 87]. Since it is a better one of the heuristics proposed we have used it. Any other heuristic can be tried in place of this. Formulation 3 will give better vote assignment but it is more complex than the other two.

Before giving the mathematical forms for each of them, we shall explain the notations used. For each formulation, first we give the mathematical forms followed by an example to clarify them.

3.2.1 Notations

N Total number of nodes in the network

k Number of groups selected by the simulator module for integer programming solution.

G_i i'th group selected by the simulator

T_i Average transaction gain in group G_i

X_i binary variable

d a large number

3.2.2 Formulation 1

Through formulation 1, we will get a vote assignment which assigns majority of votes to a subset of k groups for which sum of T_i is maximum. If there are a number of vote assignments possible it will select the one which has minimum number of total votes assigned.

As mentioned above, X_i is a binary variable. X_i will be zero if group G_i has majority of votes and X_i will be one otherwise. We let d be a large number, such that it is greater than majority of votes to be assigned. The integer programming model is:

Minimise:

$$\sum_{i=1}^{k}(T_i * X_i) \tag{3.1}$$

subject to

$$\frac{(\sum_{j=1}^{N} V_j) + 1}{2} - \sum_{k \in G_i} V_k \leq d * X_i, \ (i = 1, \ldots, k) \tag{3.2}$$

$$X_i = 0 \text{ or } 1 \ (i = 1, \ldots, k) \tag{3.3}$$

Since X_i equals zero represents the i'th group getting majority of votes, $T_i * X_i$ will represent the average number of transactions lost due to i'th group not getting majority of votes. The objective function (eq. 3.1) is trying to minimise the sum of average number of transactions lost due to the groups not getting majority of votes which is our aim. The first term in constraint (eq. 3.2) $\frac{(\sum_{j=1}^{N} V_j)+1}{2}$ represents majority of votes and the second term $\sum_{k \in G_i} V_k$ represents the votes assigned to group G_i.

case 1 When X_i is zero constraint (3.2) will become

$$\frac{(\sum_{j=1}^{N} V_j) + 1}{2} - \sum_{k \in G_i} V_k \leq 0 \tag{3.4}$$

which makes sure that i'th group will have votes greater than or equal to majority of votes.

case 2 When X_i is one constraint (3.2) becomes

$$\frac{(\sum_{j=1}^{N} V_j) + 1}{2} - \sum_{k \in G_i} V_k \leq d \tag{3.5}$$

The value of d is chosen such that when X_i is one, whatever be the votes assigned to group G_i the constraint (3.2) is still satisfied. As we can see from (eq. 3.5), the value on left hand side will be at the most majority of votes (since votes are positive). So value of d is chosen greater than the possible majority of votes to satisfy constraint (3.2). This also means that constraint (3.2) may assign majority of votes to group G_i when X_i is one. We have to make sure that whenever X_i is one the i'th group will not have majority of votes. Since the objective function is trying to minimise $T_i * X_i$ it will always try to assign all X_i's as zero. By case 1, $X_i = 0$ means G_i gets majority of votes. If it is not possible to assign majority of votes to all the groups then some X_i's will have to take the value one. The objective function will select a subset out of the k groups such that the sum of T_i for these groups is maximum and assign the corresponding X_i as zero. Constraint (3.2) will force these groups to get majority of votes. All the other groups which have corresponding X_i as one doesn't get majority of votes, since the objective function would have assigned X_i as zero if they can get majority of votes. For our experiments we have chosen d as 500.

Example:

Let us consider a 3 node fully connected network each component with a reliability of 0.9, let request arrival rates at all the nodes be 200 per unit time and let k = 3. The top 3 partitions are found to be (using simulation) {1,2,3}, {1,2} and {2,3}. The average transaction gain in each of the groups are 400, 30 and 30 respectively. The above formulation will result in:

Minimise:

$$400X_1 + 30X_2 + 30X_3$$

subject to:

$$-V_1 - V_2 - V_3 - 1000X_1 \leq -1$$

$$-V_1 - V_2 + V_3 - 1000X_2 \leq -1$$

$$V_1 - V_2 - V_3 - 1000X_3 \leq -1$$

$$X_1, X_2, X_3 = 0 \text{ or } 1$$

This will result in a vote assignment of (0,1,0)

3.2.3 Formulation 2

For vote assignment to exist a necessary (but not sufficient) condition is that all the groups should intersect with each other. The integer programming algorithm might take more time to realise this. So we can make the formulation more efficient by filtering the constraints with the above rule.

It can be shown that finding the best possible set of groups which intersect with each other is NP-complete (it is reducible to graph clique problem [KM 84]). The problems complexity is exponential in the number of groups. Since the total number of groups we are considering is constant (20), it doesn't become too much overhead. The time taken by formulation 1 to find the intersecting groups is found to be much more than time taken by this algorithm. We have implemented an algorithm which gives the best possible subset of the groups which intersect in the decreasing order of average transaction gain. We try for integer programming solution with these groups using the mathematical form shown below. If there is no solution with these groups, we try for the next best set of groups which intersect and so on (there may not be vote assignment possible for intersecting groups eg. coterie). The integer programming model is:

Minimise:

$$\sum_{i=1}^{N} V_i \tag{3.6}$$

Subject to:

$$\sum_{j \in G_i} V_j \geq \frac{(\sum_{m=1}^{N} V_m) + 1}{2}, (i = 1, \ldots, k') \tag{3.7}$$

where k' = intersecting groups of the k groups.

Since V_i represents the votes assigned to node i, objective function (eq. 3.6) is trying to minimize the total number of votes assigned. Constraint (eq. 3.7) will make sure that all the groups (intersecting groups found above), are getting majority of votes. So the above formulation will give a vote assignment, if it exist, with minimum number of votes assigned such that all the groups given to the formulation will have majority of votes. If there is no vote assignment possible, then the previous algorithm is used to get the next best set of intersecting groups. This process is continued till a vote assignment is obtained.

Example:

Let us consider the same network given in the previous example. The top groups are found to be {1,2,3}, {1,2} and {2,3}. The average transaction gain in each of the groups are 400, 30 and 30 per unit time respectively. The intersection finding module will give all the three groups as intersecting groups. The formulation will result in:

Minimise:

$$V_1 + V_2 + V_3$$

subject to:

$$V_1 + V_2 + V_3 \geq 1$$

$$V_1 + V_2 - V_3 \geq 1$$

$$-V_1 + V_2 + V_3 \geq 1$$

This will result in a vote assignment of (0,1,0).

With the above two formulations we get a vote assignment which has best subset of the top k (20) groups getting majority of votes. But there might exist a number of vote assignments possible which also can do the same. In that case the above approaches will select the one with least number of total votes assigned. In the next formulation we try to select the best one among the possible vote assignments.

3.2.4 Formulation 3

In this formulation, we use either formulation 1 or formulation 2, to find the subset (k') of the k groups for which it is possible to assign majority of votes, such that the sum of average transaction gain in these groups is maximum. The basic idea in this approach is that when the top 20 groups are unable to choose a vote assignment, we use one of the known heuristics to give weights to the nodes. Then while ensuring that the best subset k' out of k groups still get majority of votes, we try to assign votes such that the percentage of votes assigned to each node will be greater than

or equal to to corresponding percentage of these weights. By this we are trying to give priority to individual nodes in the vote assignment. If it is not possible to assign votes to all the nodes with this principle, then we will assign to the best subset of N, which is determined through the above weights. We use a modified version of heuristic proposed in [BG 87] to get the weights. This will make sure that we get majority of votes to the best subset among top 20 most likely groups and also it will hopefully give majority of votes to groups other than top 20 which also have high average transaction gain.

Heuristic 1

Weight assigned to node i (w_i) $= 100 * \lambda_i * p_i * \sum(p_k * l_j)$ for all j such that l_j is the link between nodes i and k.

Multiplication factor (100) is to reduce the error due to round off operation.

This formulation is similar to the formulation 1, where we tried to select the best subset of k groups, here we use the same idea to get the best subset of N nodes. X_i is a binary variable. X_i will be zero if node i is assigned a percentage of votes greater than or equal to percentage of its weight w_i. X_i will be one otherwise. When it is not possible to assign votes to all the nodes using the above principle, we shall try to get a best subset of the nodes such that sum of percentage of weights is maximum and all the constraints getting majority of votes in formulation 1 or formulation 2 are still having majority of votes. Let TV be the total number of votes assigned by formulation 1 or 2. The integer programming model is

Minimise:

$$\sum_{i=1}^{N}(w_i * X_i) \tag{3.8}$$

Subject to:

$$\sum_{j\in G_i} V_j \geq \frac{(\sum_{m=1}^{N} V_m)+1}{2} (i=1,\ldots,k') \tag{3.9}$$

$$\frac{w_i}{W} - \frac{V_i}{V} \leq X_i (i=1,\ldots,N) \tag{3.10}$$

where

$$W = \sum_{i=1}^{N} w_i \tag{3.11}$$

$$V = p * TV \tag{3.12}$$

We take p such that $p * TV \leq maximum(41, N)$ and $(p+1) * TV > maximum(41, N)$.

Since X_i equals zero represents the i'th node getting percentage of votes greater than or equal to to the percentage of weights assigned by Heuristic 1, $T_i * X_i$ will represent the weight of node getting percentage of votes less than its percentage of weight (w_i). The objective function (eq. 3.8) is trying to minimise the sum of weights of nodes getting percentage of votes less than its percentage of weight (w_i) which is our aim. Since Vi represents the number of votes assigned to node i, the first term in constraint (eq. 3.9) represents the votes assigned to group i and the second term represent the majority of votes in the system. So the constraint (3.9) will make sure that all the k' groups will get majority of votes. The first term in constraint (eq. 3.10) $\frac{w_i}{W}$ represent fraction of weight assigned to node i and the second term $\frac{V_i}{V}$ represents

the fraction of votes assigned to node i.

Case 1 When X_i is zero constraint 3.10 will become:

$$\frac{w_i}{W} - \frac{V_i}{V} \leq 0 \text{ (i = 1,...,N)} \tag{3.13}$$

which makes sure that i'th node will have percentage of votes greater than or equal to the percentage of its weight assigned by heuristic1.

Case 2 When X_i is one constraint, 3.10 becomes:

$$\frac{w_i}{W} - \frac{V_i}{V} \leq 1 \text{ (i = 1,...,N)} \tag{3.14}$$

From the above equation, it is clear that when X_i is one constraint (3.10) may assign percentage of votes to node i such that it is greater than or equal to percentage of its weight. We have to make sure that whenever X_i is one the i'th node will not have the required percentage vote assignment. Since the objective function is trying to minimise $w_i * X_i$ it will always try to assign all X_i's as zero. By case 1, $X_i = 0$ means node i gets required percentage vote assignment. If it is not possible to assign required percentage of votes to all the nodes then some X_i's will have to take the value one. The objective function will select a subset out of the N nodes such that the sum of w_i for these groups is maximum and assign the corresponding X_i as zero. Constraint (3.10) will force these groups to get required percentage vote assignment. All the other nodes which have corresponding X_i as one doesn't get the required vote assignment, since the objective function would have assigned X_i as zero if they can get the required percentage of votes. The value of total votes assigned through eqn.

(eq. 3.12) will make sure that if nothing can be done then this formulation will result in the same vote assignment to that of formulation 1 or 2. As value of V increases, the time taken to assign votes will also increase.

Example:

Let us consider the same network given in formulation 1 and the weights of the nodes will be (29160,29160,29160). The above formulation will result in:

Minimise:

$$29160V_1 + 29160V_2 + 29160V_3$$

Subject to:

$$V_1 + V_2 + V_3 \geq 1$$

$$V_1 + V_2 - V_3 \geq 1$$

$$-V_1 + V_2 + V_3 \geq 1$$

$$3V_1 + 123X_1 \geq 41$$

$$3V_2 + 123X_2 \geq 41$$

$$3V_3 + 123X_3 \geq 41$$

$$V_1 + V_2 + V_3 = 41$$

$$X_1, X_2, X_3 = 0 \text{ or } 1$$

This will result in a voteassignment of (14,14,13). This is a better vote assignment than obtained through the other formulations. Since all the nodes are equally important, this vote assignment will result in better performance. This formulation has tried to get vote assignment similar to the one assigned by Heuristic 1 and it even makes sure that all the groups, out of k groups, getting majority of votes in other formulations are also getting majority of votes here. It is basically able to get a vote assignment such that the groups other then k' groups getting majority of votes have more average transaction gain in them than those obtained using the other formulations.

Chapter 4

IMPLEMENTATION OF VAT

Here we describe the implementation details of VAT. This contains two major parts: simulation and integer programming. First we explain the input, output formats of the VAT and then we describe the functions of the important modules present in both the parts.

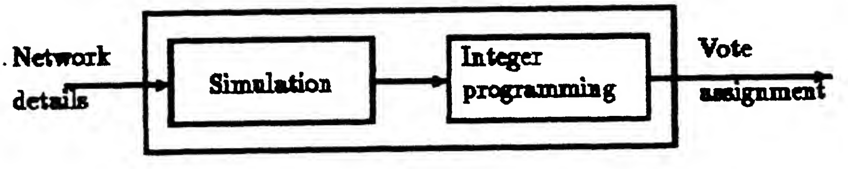


Figure Approach

We assume that the mean time to repair (MTTR) of each component is 240 time

units. Mean time to failure (MTTF) is calculated using reliability of the component as:

$$\text{reliability} = \frac{MTTF}{MTTF+MTTR}$$

The simulation is done for a period of 10000000 time units.

4.1 Input format

The input file contains the following information:

- Format of the output (normal = 0, Verbose = 1).
- File containing information on nodes.
- File containing information on links.

A sample input file is shown below:

```
1
nodes.1
links.1
```

The input file containing information about nodes has triplets of the form:

< node#, rel, req_arr_rate > where:

node#	number of the node
rel	reliability of that node
req_arr_rate	request arrival rate at that node

A data file on nodes (nodes.1) is shown below:

1	0.9	1.0
2	0.9	1.0
3	0.9	1.0

The input file containing information about links has triplets of the form:

< nd1, nd2, rel > where:

nd1,nd2	The end points of the corresponding link
rel	Reliability of the link

Since the links are bidirectional there will be only one entry for a link between any pair of nodes. A data file on links (links.1) is shown below:

1	2	0.9
1	3	0.9
2	3	0.9

4.2 Output format

Output is given in two modes. One of the two modes, normal or verbose, is selected in the input file.

4.2.1 Normal mode

In the normal mode, the output will show the votes assigned by formulation 2 and then it will show the final vote assignment obtained using formulation 3.

For the above input data, the VAT's output in this mode is:

Vote assignment by

FORMULATION 2

VOTE[1] = 0

VOTE[2] = 1

VOTE[3] = 0

Final vote assignment

FORMULATION 3

VOTE[1] = 14

VOTE[2] = 14

VOTE[3] = 13

4.2.2 Verbose mode

In this mode, in addition to the vote assignment it will show the top k groups obtained through simulation and the average number of transactions gained in each of them. It will also show the groups getting majority of votes through the vote assignment.

For the input file shown in 4.1, VAT's output is:

Top k groups

Group	Average transaction gain
{1,2,3}	400
{1,2}	30
{2,3}	30

Groups getting majority of votes

Group	Average transaction gain
{1,2,3}	400
{1,2}	30
{2,3}	30

Vote assignment using

FORMULATION 2

VOTE[1] = 0

VOTE[2] = 1

VOTE[3] = 0

Final vote assignment using

FORMULATION 3

VOTE[1] = 14

VOTE[2] = 14

VOTE[3] = 13

4.3 Simulation model

We have used an event driven simulator model to find the most likely groups formed due to partition failures along with the average transaction gain in each of them, if they were assigned majority of votes. The simulation data is obtained during run

time. ie. the next failure time of any node is calculated only after the current failure of that node, similarly for the other events. With this there is no need to store the past or future event times, which reduces the memory requirement of the simulation program. The basic modules present here are:

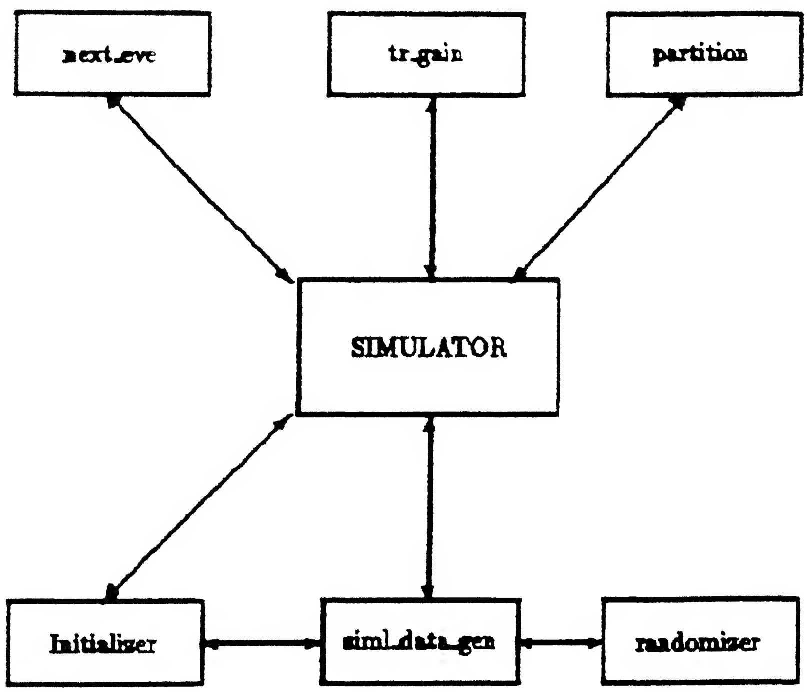


Figure Simulation modules and their interactions

Initialiser: It initializes the network topology and gets the times at which failure of each component occurs, for the first time using the given input data.

Siml_data_gen: This is used to get the time at which next event occurs at the same node/link. We have assumed that the node/link failure/recovery times are exponen-

tially distributed, the time at which next event occurs is calculated as

$$T_n = T_{n-1} + (-m\_t * ln(k))$$

where

T_{n-1} = time at which the event has occured.

m_t = mean inter event time.

k = pseudo-random number between 0 and 1.

Next_eve_fnd : This is used to find the time at which the next immediate global event occurs.

Partition_finder: This module keeps track of the network topology. Whenever there is a failure/recovery, it modifies the network topology accordingly, finds the connected components and gets the top two groups present in the network by finding TR_GAIN_RATE of different groups.

tr_gain_fnd: This module keeps track of the HPG/NHPG groups formed and finds the total fraction of time for which each group is present as HPG/NHPG. We kept track of the first 4000 HPG/NHPG groups formed during simulation.

Simulator: This is the main module. It initially gets the input data for simulation and makes a call to initialiser module to start the simulation. It uses Next_eve_fnd module to get the time at which next event occurs and increments the simulation time to it. Whenever a failure/recovery occurs, it makes a call to partition_finder

module and uses tr_gain_fnd to keep track of the groups. Finally, it gets the top k groups obtained using AVG_TR_GAIN of each of the groups.

4.4 Integer programming

This module gets the top k (k = 20) groups along with the average transaction gain in each of the groups, if they were assigned majority of votes from simulation part. Its aim is to get the vote assignment such that the average transactions gained in the system is maximized. We use formulation 3 to get the vote assignment since it gives better vote assignment than the other two. It has three major modules: prob_formulator, IP_routine, vote_assign.

prob_formulator: It uses the groups given by the simulator module and formulates this as an integer programming problem, using one of formulation 1 or 2 or 3. Here, we have used formulation 3, in which formulation 2 is used to find the k' groups which should get majority of votes.

IP_routine: This uses Gomory's method with Wilson's cuts algorithm [WS 67] to solve the integer programming problem.

vote_assign: This is the main module in this part. Initially it makes a call to prob_formulator module and gets the integer programming problem for formulation 2. With this, it uses IP_routine to get the best subset out of k groups which will get majority of votes. Using these groups it again calls prob_formulator to gets the integer programming problem using formulation 3 and then calls IP_routine to get

the final vote assignment.

4.5 Random graph generator

We have used the model suggested in [BW 88], where it is claimed that the graphs generated here will have characteristics of real time networks. We have implemented this model to get the random graphs and obtained the vote assignment using VAT.

All the integer programming formulations are verified on the standard commercially available LiNear Discrete Optimiser (LINDO) package.

Chapter 5

EXPERIMENTS

We have tried the vote assignment tool (VAT) for different types of networks. In this section, we discuss the experiments done using VAT. As mentioned before, VAT considers the average number of transactions gain per unit time as the performance metric. It was found that in many of the cases VAT is assigning votes similar to those assigned by the best one of the existing heuristics. These heuristics have different metrics for performance evaluation. One of the metric used is steady state probability that the system is up.

5.1 Experiment 1

We have used VAT to find the vote assignment, for the different network configurations given in [BG 87], [TK 91]. The following table shows the results. We have

calculated the best vote assignment done by the heuristics for these networks, the equivalent vote assignment with minimum number of total votes assigned is obtained and it is given in the following table. (Vote assignments are said to be equivalent if the groups getting majority of votes are same in all of them.) The table also gives the vote assignment with minimum total votes assigned and which is equivalent to that of VAT's assignment.

Network configurations

5 node fully connected network

1. All components reliability = 0.6
2. All components reliability = 0.7
3. All components reliability = 0.8
4. All components reliability = 0.9
5. All components except node a, reliability = 0.9, node a reliability = 0.95
6. All components except node a, reliability = 0.9, node a reliability = 0.98
7. All components except node a, reliability = 0.9, node a reliability = 0.99
8. All components except node a, reliability = 0.9, node a reliability = 0.999
9. Nodes a,b,c reliability = 0.9,rest = 0.5, links (a,b),(a,c)(b,c) rel = 0.9, rest = 0.5
10. Nodes a,b,c reliability = 0.9,rest = 0.5, links (a,b),(a,c)(b,c) rel = 0.9, rest = 0.8

11. Nodes a,b,c reliability = 0.9,rest = 0.8, links (a,b),(a,c)(b,c) rel = 0.9, rest = 0.8

12. Node a reliability = 0.99,rest = 0.8 all links = 0.9

13. Node a reliability = 0.999,rest = 0.8 all links = 0.8

14. Node a,reliability = 0.999,rest = 0.85 all links = 0.85

<u>6 node fully connected network</u>

15. All components reliability = 0.6

16. Nodes a,b,c rel = 0.9,rest = 0.6, links among a,b,c = 0.9 rest = 0.6.

17. Nodes a,b,c rel = 0.9,rest = 0.6, links among a,b,c = 0.9 rest = 0.5.

18. Node a reliability = 0.9,rest = 0.6, links reliability = 0.6

19. All components reliability = 0.9

<u>Nonfully connected 5-node system</u>

20. same as 6), but link (a,e) deleted.

21. same as 20), but links (a,b) and (c,d) are deleted.

22. same as 21), but links (c,e) and (b,d) are deleted.

The vote assignment for the last two cases is given by the heuristic given in [TK 91] where it is assumed that the reliability obtained by their method will be compared with the reliabilty given by the singleton and uniform voting and then the best one is selected. This approach is not feasible for big networks since finding the probability that a group of nodes which remain connected is NP-hard [AR 77].

Vote assignment			
Case	Best assignment by the heuristics	VAT's assignment	VAT's equivalent minimal assignment
1	(1,0,0,0,0)	(9,8,8,8,8)	(1,1,1,1,1)
2	(1,1,1,1,1)	(9,8,8,8,8)	(1,1,1,1,1)
3	(1,1,1,1,1)	(9,8,8,8,8)	(1,1,1,1,1)
4	(1,1,1,1,1)	(9,8,8,8,8)	(1,1,1,1,1)
5	(1,1,1,1,1)	(9,8,8,8,8)	(1,1,1,1,1)
6	(1,1,1,1,1)	(9,8,8,8,8)	(1,1,1,1,1)
7	(3,1,1,1,1)	(9,8,1,8,9)	(2,1,1,1,2)
8	(1,1,1,1,1)	(17,1,7,1,9)	(3,1,1,1,1)
9	(1,1,1,0,0)	(10,10,11,4,4)	(1,1,1,0,0)
10	(1,1,1,0,0)	(14,9,11,5,0)	(1,1,1,0,0)
11	(1,1,1,1,1)	(9,9,9,7,7)	(1,1,1,1,1)
12	(3,1,1,1,1)	(17,7,1,9,1)	(3,1,1,1,1)
13	(1,0,0,0,0)	(25,8,0,0,8)	(1,0,0,0,0)
14	(1,0,0,0,0)	(25,0,8,8,0)	(1,0,0,0,0)
15	(1,0,0,0,0,0)	(7,7,7,7,7,6)	(1,1,1,1,1,0)
16	(2,2,2,1,1,1)	(8,8,8,5,5,5)	(2,2,2,1,1,1)
17	(1,1,1,0,0,0)	(9,9,9,4,4,4)	(2,2,2,1,1,1)
18	(1,0,0,0,0,0)	(1,0,0,0,0,0)	(1,0,0,0,0,0)
19	(0,1,1,1,1,1)	(6,7,7,7,7,7)	(0,1,1,1,1,1)
20	(1,1,1,1,1)	(7,9,9,9,7)	(1,1,1,1,1)
21	(1,0,0,0,0)	(6,8,9,9,9)	(1,1,1,1,1)
22	(1,0,0,0,0)	(16,7,8,8,0)	(2,1,1,1,0)

Table Vote asignments

It was observed that in many of the cases the vote assignment done by VAT is equivalent to that of the best assignment given by the heuristics. Since the metric used for performance evaluation is different, in some of the cases the vote assignment done by VAT is different to that of the best vote assignment done by the heuristics.

5.2 Experiment 2

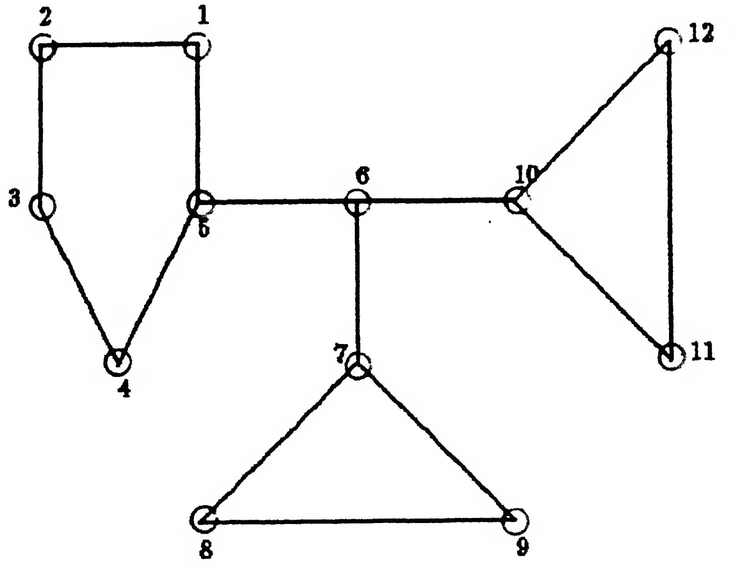
Here we have obtained the random graphs using model suggested in [BW 88]. From this model, we get the topology of the network. The reliabilities of each of the components is assigned such that all the reliabilities are greater than 0.5. The networks for different connectivities and for different number of nodes are obtained using the above model. We have obtained vote assignment for these networks using VAT.

In all the existing heuristics the number of votes assigned to each node depends upon the reliability of the neighbouring nodes and the reliability of the links incident on it. But there exist cases in which the priority of the node should be determined not only on the local information but also on how well it is connected to all the other nodes and reliability of all the nodes. Let us consider the following network.

- All nodes reliability, except node 6 = 0.9
- Node 6 reliability = 0.6
- Reliability of links (5,6),(6,7),(6,10) = 0.9
- Reliability of other links = 0.95

All the existing heuristics will give equal priority to the nodes 1, 8 and 11, eventhough node 1 is present in a better partition. Our tool determines through simulation, that most likely groups are {1,2,3,4,5}, {7,8,9} and {10,11,12}. Since {1,2,3,4,5} is a better group it assigns majority of votes to this group.

Evaluation of vote assignments is not feasible for big networks. So, for big networks, it is not feasible to compare the performance offered by VAT's vote assignment with other heuristics. Since global information will be more important in these type of networks, VAT's vote assignment will give better performance than the other heuristics.



Chapter 6

CONCLUSIONS

Voting is a well known technique for controlling access to replicated data. The votes assigned to each of the nodes can have a remarked effect on the system performance. In this report, we have discussed an integer programming approach to solve the vote assignment problem using the average transaction gain per unit time as the performance metric. In this approach, first we use monte-carlo simulation to determine the priority groups and then using these groups the vote assignment problem is formulated as an integer programming problem. The entire procedure has been implemented in the Vote Assignment Tool (VAT).

Unlike the existing heursistics, our approach uses global information in deciding voting pattern ie. the votes assigned to each of the nodes depend upon how well it is connected to all the other nodes and reliability of all the nodes. This has resulted in a better voting pattern. But the main drawback is as the number of nodes increases,

the overhead involved also increases. We have tried the tool on the test cases given in [BG 87], [PK 91]. In most of the cases it is assigning votes similar to the best one of the heuristics. We have also used this tool on the random graphs generated by the model suggested in [BW 88], to determine the vote assignments.

REFERENCES

[AD 76] P.A.Alsberg, J.D.Day, *A principle for resilient sharing of distributed resources*, In Proc. 2nd Int. Conf. software engg. ,1976, pp562-570.

[AR 77] A.Rosenthal *Computing the reliability of a complex network*, SIAM Journal on Applied Mathematics, Vol.32, 1977,pp384-393.

[BG 84] D.Barbara, H.Garcia-Molina, *The vulnerability of voting mechanisms*, In proceedings 4th Symp. Reliability Distributed Software Database Syst. , October 1984, pp45-53.

[BGA 86] D.Barbara,H.Garcia-Molina,A.Spauster, *Policies for dynamic vote reassignment*, Proc. IEEE Conf. on Distributed Computing, May 1986, pp195-206.

[BG 87] D.Barbara,H.Garcia-Molina, *The Reliability of Voting Mechanisms*, IEEE Trans. comput. Vol. C-36, No. 10,October 1987, pp1197-1207.

[BH 87] A.Bernstein,V.Hadzilacos,N.Goodman, *Concurrency control and recovery in Database systems*, Addison-Wesley, 1987.

[BW 88] B.M.Waxman, *Routing of multipoint connections*, IEEE Journal on selected areas in commn. , Vol. 6,No. 9, December 88, pp1617-1622.

[CD 88] G.F.Coulouris, J.Dollimore, *Distributed Systems concepts and design*, Addison-wesley Publishing Company, 1988.

[CMM 90] S.Y.Cheung,M.Ahamad,M.H.Ammar, *Multi-Dimensional Voting: A general method for implementing synchronization in distributed systems*, Proc. of the Int. Conf. on Distr. Computing Sys. June 1990.

[DD 89] D.Davcev, *A Dynamic voting scheme in Distributed systems*, IEEE Trans. on Software Engg. Vol. 15, No. 1, Jan 1989,pp93-97.

[DGB 85] S.B.Davidson,H.Garcia-Molina,D.Skeen, *Consistency in partitioned networks*, ACM computing surveys, Vol. 17, No. 3, September 1985,pp341-370.

[GB 85] H.Garcia-Molina, D.Barbara, *How to assign votes in a Distributed system*, Journal of the ACM, Vol. 32, No. 4, October 1985, pp841-860.

[GI 79] D.Gifford, *Weighted Voting for replicated data*, In proceedings of the seventh symposium on Operating Systems Principles, ACM SIGOPS, December 1979, pp150-159.

[HG 82] H.Garcia-Molina, *Elections in a Distributed Computing System*, IEEE Trans. computers, Vol. C-31, No. 1, Jan. 1982, pp48-59.

[HS 89] H.M.Salkin, K.Mathur, *Foundations of integer programming*, North-Holland,1989.

[JM 87] S.Jajodia,D.Mutchler, *Dynamic voting*, Proc. of SIGMOD 87,pp227-238.

[JP 86] J. Paris, *Voting with witnesses: A consistency scheme for replicated files*, In Proc. 6th Int. Conf. Distributed Computing Systems, May 1986, pp606-612.

[KM 84] K.Mehlhorn, *Data structures and Algorithms 2: Graph Algorithms and NP-Completeness*, EATCS Monographs in Theoritical computer Sc. , Springer Verlag,1984.

[LSP 82] L.Lamport,R.Shostak,M.Pease, *The byzantine generals problem*, ACM Trans. Prog. lang. Syst. Vol. 4,No. 3,July 1982,pp382-401.

[MK 85] M.Maekawa, *A $\sqrt{N}$ Algorithm for mutual exclusion in decentralized systems*, ACM Trans. Computer systems, May 1985, pp145-159.

[MM 89] Mustaque Ahamad, Mostafa H. Ammar, *Performance characterization of Quorum-Consensus Algorithms for Replicated Data*, IEEE Trans. Software Engg. , Vol. 15, No. 4, April 1989, pp161-168.

[ND 79] Narsingh Deo, *System simulation with digital computer*, Prentice Hall of India pvt. ltd. , 1979.

[NP 66] N.P.Buslenko, D.I.Golenko, Yu.A.Screider,I.M.Sobol, V.G.Sragovich, *The Monte carlo method: The method of statistical trials*, Pergamon press, 1966.

[OB 90] M. Obradovic, P.Berman, *Voting as the static pessimistic scheme for managing replicated data*, 9th IEEE Symp. on Reliable Distributed Systems,1990.

[PB 85] K.P.Birman, *Replication and Fault-tolerance in the ISIS System*, 10th ACM Symp. on O. S. pri. ,Dec 1985.

[PN 86] C.Pu, J.D.Noe,A.Proudfort, *Replication in distributed systems: The Eden Experience*, Proc. Fall joint computer conf. 1986,pp1197-1209.

[PR 83] D.S.Parkar et. al. *Detection and resolution of inconsistencies in distributed systems*, IEEE Trans. Software Engg. May 1983, pp342-349.

[RT 88] R.Renesse, A.Tanenbaum, *Voting with Ghosts*, Int. Conf. on Distributed Computing Systems, 1988.

[SK 91] S.Rangarajan,S.Setia,S.K.Tripathi, *Fault-tolerant Algorithms for Replicated data management*, Technical Report, Institute for Advanced Computer Studies, University of Maryland, June 1991.

[SS 83] R.D.Schlichting and F.B.Schneider, *Fail-stop processors: an approach to designing fault-tolerant computing systems*, ACM Trans. Comput. sys. Vol. 1,1983,pp222-238.

[TK 91] Z.Tong, R.Y.Kain, *Vote assignments in weighted voting mechanisms*, IEEE Trans. computers,Vol. 40,No. 5, May 1991, pp664-667.

[TH 79] R.H.Thomas, *A majority consensus approach to concurrency control*, ACM Trans. Database Systems,Vol. 4,June 1979, pp180-209.

[TN 89] J.Tang,N.Natarajan, *A static pessimistic scheme for handling replicated databases*, ACM SIGMOD Int. conf. on management of data,1989, pp389-398.

[WS 67] R.B.Wilson *Stronger cuts in Gomory's all integer programming algorithm*, Operations research, Vol 15, 1967, pp155-156.